THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

An Experimental Teleterminal—The Software Strategy

By D. L. BAYER and R. A. THOMPSON

(Manuscript received September 29, 1982)

A research model of a synergism between a telephone and a computer terminal is described in this paper and its companion. The hardware design of this "teleterminal," described in the companion paper, includes an internal microprocessor and a data connection to a host computer. This paper describes the software in these two machines. The software resident in the teleterminal's internal microprocessor addresses internal issues like cursor control, tab expansion, and data modes. The software resident in the host controls screen content and menu selection. The teleterminal is a research tool for the investigation of the human interface for the access to experimental services. Such services include calling by name, directory retrieval, a variety of information (e.g., yellow pages, department store catalogs, community bulletin boards, newspapers, and libraries) and a variety of personal services such as mail, personal calendar, entertainment, and shopping. The user-interface is a conceptual tree-like structure that can be customized by the user.

I. INTRODUCTION

A *teleterminal* is defined to be a piece of equipment that merges the functionality of the traditional *tele*phone with that of a computer *terminal*. It is characterized by:

(*i*) A traditional telephone facility
(*ii*) Internal intelligence
(*iii*) A data communication facility
(*iv*) A general-purpose display
(*v*) Dynamic labeling of buttons (soft keys).

This merger of functionality has been seen to be synergistic. One such teleterminal, whose hardware is described in Ref. 1 and in the companion paper,[2] is illustrated in Fig. 1.

Fig. 1—A teleterminal.

The construction of and, more significantly, the experimentation with such a teleterminal is part of a large-scale, long-range research investigation into the systems, software, and applications aspects of telecommunications. Cognitive and social implications are a significant part of the study. An evolving futuristic "test-bed" environment[3] currently consists of a highly reliable host computer (a three-processor Tandem-16), an "intelligent" digital switching office,[4] and a collection of teleterminals. A number of systems-level principles have been uncovered in the course of this research and appropriate papers have been submitted to and published in appropriate journals. It is the purpose of this paper and its companion to describe the equipment used in this research and thereby serve as a common reference for those other papers of a more general nature.

Compatibility with today's environment would be of major importance if a real product were being described, but it is of little consequence in a research environment, except that it simplifies the dissemination of teleterminals to interested people. There are about three dozen of these teleterminals in an active user community. It must be emphasized that the teleterminal is *not* a proposed product; it is a research tool. Design decisions, and discussions in this paper about it, are based on this important point. If a product were anticipated, many decisions would have been made differently and the reasons discussed would change dramatically. Furthermore, neither this research nor this paper is intended to describe nor imply the future direction of the Bell System as it relates to advanced communications systems and services, nor is it intended to endorse any hardware or software offerings by any vendor.

The next section of this paper presents the research objectives of the project that have an impact on software. They include assumptions about the system environment and the impact on the user. In Section III, intelligence distribution is discussed and the internal and host programs are described. In Section IV, a scenario is presented to illustrate program execution, the user interface, and the kinds of capabilities provided. Section V contains a discussion of lessons learned and future directions.

II. OBJECTIVES

Objectives that have software implications are emphasized. These are presented from the viewpoint of both the system and the user.

2.1 System environment

In the current system configuration, each teleterminal has a standard terminal connection to a host computer and a standard telephone connection to a telephone office. The only assumed commonality

occurs at the teleterminal itself. Compatibility with the "POTS" (plain old telephone service) world requires that old-fashioned telephone hardware like the switchhook and *Touch-Tone** dialing equipment be necessary parts of the teleterminal.

Such a system configuration impacts software in that this required POTS hardware reduces available space for microprocessor memory inside the teleterminal and that special microprocessor software is needed to interact with this POTS hardware. Furthermore, the host software must transmit phone numbers to the teleterminal (with a special data mode) instead of directly to the telephone office controller and it must be concerned with such POTS functions as the switchhook state and dial tone. "POTS" functions, like "busy" or "network blocking", could be recognized by including certain tone-detecting filter circuits in the hardware, but to conserve space this was not done. As a consequence, the software is limited to assuming the presence of dial tone after a fixed delay and the software can make no branches on whether or not a call completes.

A second system aspect is our desire to use the teleterminals in a "test-bed" environment for the ongoing investigation of the human interface and the study of experimental services. This suggests that changes will be made to the software frequently and reflection of this in the initial design of the software architecture was felt to be wise. A distribution of intelligence was selected wherein the functional software is located in a centralized general-purpose host computer. Furthermore, this "access program" can be structured to simplify anticipated changes. Such a "host-centered" distribution of intelligence may not be ideal for a marketed product where the design goals would be different and the interface would be more stable. These kinds of items are discussed in Section V.

2.2 User Impact

After system environment, the second area of objectives with software ramifications is the impact on the user. There are two such impacts: the physical cosmetics of the teleterminal and the conceptual interface to the software. Cosmetic features that affect software are the small size, the incorporation of both telephone and terminal characteristics, and the inclusion of function keys.

For the experiments with new applications, the teleterminal replaces a conventional business telephone. It is important that the set occupy approximately the same desk space as a conventional business telephone. Furthermore, the set is "real" in the sense that all electronics, except power supplies, reside within the set. This results in a limited

* Registered service mark of AT&T.

physical space for internal memory and thus restricts the size of the internal software. The placement of the function keys adjacent to the cathode ray tube (CRT) screen has major impact on the software. At the "low" level, internally executed "primitives" are required for labeling buttons and sensing button-pushes. At the "high" level, these buttons make possible the use of a tree-structured user interface to access various functions.[5]

From the user view, three general requirements of this access method are simplicity, convenience, and customizability. The interface must be as conceptually simple as possible, even to the most casual, unsophisticated user. The required convenience of the interface is attained by enhancing the access to often-used functions and the names of often-called people and by providing a "translation" capability between the user's name for some function and the "telephonese" or "computerese" for accessing it. Whether customization is done by the vendor or by the user, the need for real simplicity in the underlying data structure, and not just apparent simplicity in the user's view of it, is reinforced.

The structure of the user interface is tree-like in a deliberate attempt at "congruency"[5] with simple models of human cognition.[6] Similar "access trees" have appeared in the literature. References 7 and 8 are representative. In the marketplace, Hewlett-Packard's new line of terminals and the British PRESTEL are examples. In the mathematical sense, a "tree" is an acyclic, connected graph;[9] but the concept of tree to be presented is more like the "lay" notion of a large woody plant.

Informally, the parts of a tree include the root, branches, intermediate nodes, and leaves. Let a screen of button labels correspond to a node. Let the dynamic label of a button correspond to a branch or leaf, depending on whether its selection causes a traversal to a new node or the actuation of some function, respectively. The structure is not a mathematical tree because multiple branches to the same node are permitted, as are "backup" and "restart" branches (which make the "graph" cyclic). Analogous to the root, or initial node, of the tree is an initial labeling of the buttons by which the user perceives the "entering" of his structure of functions and directories. The root of an example access tree is illustrated in Fig. 2.

The organization of this access tree is intended to be defined or selected by the user. A *functional* organization would show a root menu with branches like **Telephone Functions,*** **Computer Functions, Calendar, Mail,** etc. A utilitarian organization would attempt to place highly used functions near the root and seldom used functions

* Labels identifying function keys are set in boldface type.

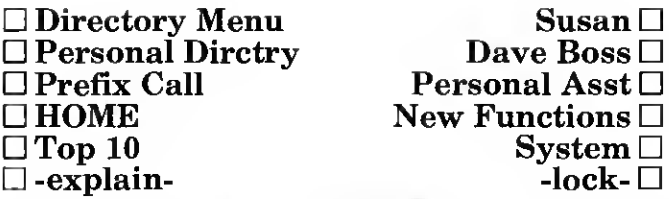

Fig. 2—A sample access tree.

far from the root. Some compromise of the two organizations is probably most appropriate, depending on the user's level of sophistication and the frequency of use.

III. INTELLIGENCE DISTRIBUTION

The user interface to system capabilities is determined by the properties of the terminal and the network resources that the user can access through the terminal. The capabilities of the terminal are constrained by the computing power of the internal processor and by the data switching capabilities of the communications system. Currently available microprocessors span at least an order of magnitude in speed and several orders of magnitude in memory size. In the case of the teleterminal, power requirements and chip count were important considerations in the selection of the internal microprocessor. Many of the design considerations described in the previous section constrained the type of internal computations and thus focused our efforts on software issues relating to telephone applications vis-a-vis general-purpose word-processing systems.

In the next subsection, the capabilities of the processor within the teleterminal are presented. A list of functions performed by the instrument is given and the impact of this processing power on feature software is discussed. The limitations of, and alternatives to, the current design are presented in Section V.

3.1 Internal processing

The teleterminal contains an Intel 8748 microprocessor with 2048 bytes of programmable read-only memory (PROM) for program storage and 288 bytes of random access memory (RAM) for data storage. The processor executes instructions in from four to eight microseconds. A common operation such as moving a byte from one location to another in RAM requires four instructions, occupies six bytes of program memory, and takes 32 microseconds. A single vectored interrupt is available for performing time-critical functions.

The internal program is written in assembly language and occupies most of the available program memory. The function of the internal program is to control the microprocessor's peripherals: a Matrox

display processor, a tone generator with muting relay, an ASCII keyboard with clicker relay, and a universal asynchronous receiver/transmitter (UART) for interface to the host computer.

The Matrox display processor cycles through a 512-byte dual-ported buffer memory displaying its contents as 16 rows of 32 characters on a CRT. In addition to character generation, the display processor provides character blinking at a half-hertz rate. The internal processor manages the contents of the buffer memory. Scrolling is implemented by moving the entire contents of the display buffer. A cursor is implemented by displaying an underscore character (_) when a space character lies at the cursor coordinates or by blinking the character at the cursor coordinates (inverse video would be more attractive but is not available in the current hardware). Button labels are cleared, positioned, and formatted by the internal processor.

The tone generator is used to dial over the telephone line associated with the teleterminal. The state of the switchhook is examined to determine if the receiver is off-hook. After a short delay to allow for dial tone, digits are dialed by activating the muting relay, activating the tone generator for approximately 100 milliseconds, then deactivating the muting relay, and waiting approximately 60 milliseconds before repeating.

The keyboard peripheral consists of three types of keys: standard keys that correspond to ASCII codes, special control characters within the control code set that correspond to the buttons adjacent to the screen, and polled control buttons. The first two types of keys are scanned, encoded into ASCII character codes, and latched by the keyboard circuitry.[2] The processor examines the latch approximately 60 times per second. When a character has been latched by the hardware, the processor reads the character, clears the latch, and activates the clicker relay to give the user audio feedback. Normal character codes are delivered to the output UART for transmission to the host. For codes corresponding to screen buttons, the processor emits a unique three-character sequence for each of the buttons.

Control buttons are wired directly to individual bits on one of the processor's input/output (I/O) ports. After debouncing, the buttons are used to (also see Section 3.3):

(*i*) Freeze the screen

(*ii*) Emit a special host-specified "delete previous character" code

(*iii*) Emit a special host-specified "delete current line" code

(*iv*) Emit a special host-specified "interrupt" code. The internal buffer of input characters from the host is also purged.

The input characters from the UART (data from the host) are read at interrupt level and collected in a 255-byte circular buffer. In full-duplex operation, flow control is implemented by emitting a single

symbol when the input buffer approaches overflow. This symbol can be set by the host with a control message. The characters in the circular buffer are processed by the base-level part of the control program. Data is separated into three classes: control messages, response messages, and ordinary text. Control messages change the "state" of the set and determine how ordinary text is to be processed. The following "states" exist:

(*i*) Label button—Data is positioned adjacent to the button, left or right justified, and constrained to fifteen characters per line.

(*ii*) No scroll—Data is constrained to the bottom line of the screen.

(*iii*) Dial—Data is interpreted as dial control: that is, a digit to dial, a request for dial delay (usually for second dial tone), a request to time and display call duration, or a request for current call status.

(*iv*) Terminal—Data is displayed much the same as on any simple CRT terminal.

Response messages are character strings generated by the teleterminal in response to either a button push or a host request for status. These messages appear in the teleterminal's input stream because the host computer, in full-duplex mode, echos all characters transmitted from the set. Status response messages are ignored by the set. Button-push response messages cause the set to start flashing the first character of the associated button label, providing a feedback cue equivalent to echoing ordinary characters.

In summary, the internal processing capabilities are used to control the CRT display and perform dialing functions. Support for telephone-related applications is manifested in functions that dial, label buttons, flash-button labels, and restrict output to the bottom line on the CRT.

3.2 Host processing

The description of the host software is contained in the next three subsections. First the host's software environment is discussed, then the structure of the user's data is defined, and finally the "access program" is described.

3.2.1 Host software environment

At the system level, the host software environment is that of a transaction-oriented, time-sharing, general-purpose computer. The current implementation, under the *UNIX** operating system,[10] provides a convenient machinery for interfacing the access program not only to the standard commands, system calls, and supported programs of the *UNIX* operating system, such as those providing computer mail,

* Trademark of Bell Laboratories.

calendar functions, and various games, but also to current and future specialized application programs, such as have been written for mail access, call-memos, and personal calendar maintenance. The access program and the peripheral programs have been written in the C programming language[11] by a multiplicity of authors with program interfaces provided by the standard system calls and interprocess messages of the *UNIX* operating system. The complete environment resides in a Tandem computer system.

The user's tree-structured data are isolated from the access program that manages those data. This isolation is accomplished by keeping user-specific data in a separate file (per-user) and out of the program. This isolation permits the existence of a single copy of the program in a computer that supports multiple users, each with a unique and customized access structure. Furthermore, this software practice permits changes in directories or function procedures and similar modifications and customizations without requiring the recompiling of a program.

A data change to the access tree data file may be as trivial as an updated telephone number or name of an application program, or it may be purely cosmetic like renaming a button label, moving a familiar function to a new position in the tree, or placing a new label on some vacant button. Or, it may be more significant, like pruning the tree, attaching a new node to the tree, grafting an entire subtree to a new position, or adopting an entirely new access tree. Currently, these changes are made through the *UNIX* text editor to the data only and the program is not recompiled. An experimental editor with a potentially improved human interface is described in Ref. 12.

3.2.2 Data description

The access tree is logically organized as a hierarchical data structure. The current implementation uses a magnetic disk medium and so the physical organization is linearized into a serial file of records. The data consist of an iteration of "record-groups" (each corresponds to a node of the tree or screen on the teleterminal), each in turn made up of ordered "records" (each corresponds to a button label). The actual data file read by the access program is obtained by "compiling" the file to be described here. A structure diagram of the data, in the sense of Michael Jackson,[13] is shown in Fig. 3.

The label is the text that appears adjacent to the corresponding button when the parent-node is active. The successor-node identifies the node to which the tree traverses when the corresponding branch is selected. The button-number is usually omitted but, if specified, is an integer between zero and eleven. The default is the "next" button in logical order. The command is executed by the program as described

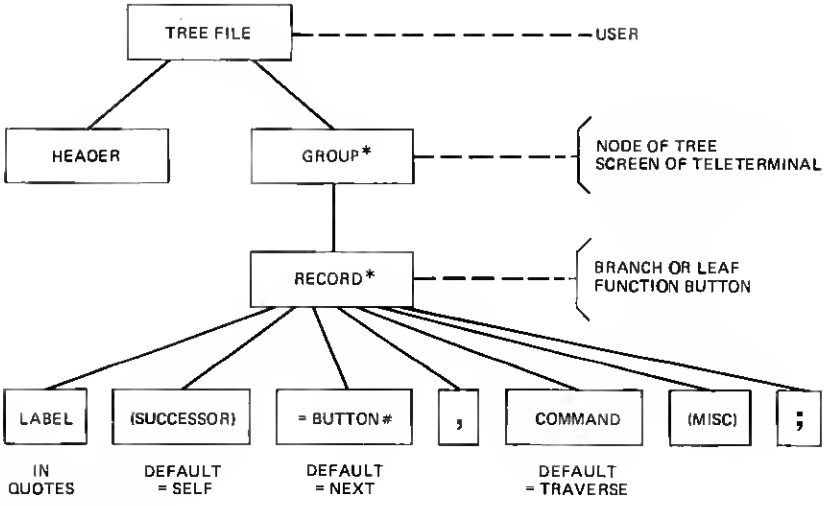

Fig. 3—Structure diagram of access tree data.

in the next subsection. The miscellaneous field contains data pertinent to the command, such as a telephone number or program name.

Data files in the format of such access trees also require "headers". Since each user is assumed to possess a personal tree file, it is appropriate to place user information in the header such as: a list of telephone numbers from which one is selected for "call-return" messages, the password for allowing access to "private" parts of the tree, and the default-assumed location used for adjusting call prefixes. The first several lines of a typical user's tree file are illustrated here in Fig. 4.

Record-groups are enclosed in brackets,"{ }", and symbolically named. The records are assigned to buttons in consecutive order within a record-group by default. The default successor-node is the current node and the default command is TRAVERSE. The correlation between a record-group and the user's perception on the CRT screen is seen by comparing the record-group named ROOT from the partial file of Fig. 4 with the "Root node" illustrated in Fig. 2. In the node corresponding to the PREROOT record-group of Fig. 4, buttons numbered one, seven, nine, and ten are left unlabeled.

From the header of the file in Fig. 4 it is seen that this tree belongs to a user whose Bell Laboratories phone number is 6170 at Murray Hill (represented by the mnemonic prefix "MH") and whose home phone number is 4649999 in dial area 201 (represented by the mne-

```
TREE beading
Numbers: murray-hill: MH6170, home: N4649999
Password: joesentme
Location: MH
%%
PREROOT
{
"introduction", CATFILE(expl/e.0,trav);
"Other User" =2; CATFILE(/get/dump/othus,exit);
"Other Pbone", OTHERPHONE;
"Other Location", OTHERLOC;
"-explain-", EXPLAIN;
"open"(ROOT), CHANGELOCK;
"Bell logo" =8, CATFILE(/get/dump/bell,trav);
"-exit-" =11, CATFILE(none,exit);
}
ROOT
{
"Directory Menu" (DTORYMENU);
"Personal Dirctry" (PERSDIRY);
"Prefix Call" (PREFIX);
"HOME", CALL(N4649999,zhome);
"Top 10" (TOP10);
"-explain-", EXPLAIN;
"Susan"(PERSASST), CALL(MH4236,sst);
"Dave Boss", CALL(MH4235,db);
"Personal Asst"(PERSASST);
"New Functions"(NEWSVC);
"System", RUNSCROLL(/bin/sb,sh);
"-lock-"(PREROOT), CHANGELOCK;
}
PERSASST
{
"Today's Appnts", RUNSCROLL(/usr/lbin/caltoday,caltoday);
"Other Appnts", RUNSCROLL(/usr/lbin/calexam,calexam);
"Make Appnts", RUNSCROLL(/usr/lbin/calenter,calenter);
"Set Reminder", REMINDER;
"Time & Date", RUNBOTTOMLINE(/bin/date,date);
"2-month Cal", CAL2MONTH;
"Read Mail"(PREFIX), READMAIL;
"Send Mail", SENDMAIL;
"Send Call-Memo", SENDCALLMEMO;
"Std. Call-Memo", STDCALLMEMO;
"-backup-"(ROOT), TEMPBACKUP;
"-restart-"(ROOT);
}
DTORYMENU
{
"Emergency"(EMERGDTORY);
  .
  .
  .
```

Fig. 4—Example of a partial tree file.

monic prefix "N"). This user's password is "joesentme" and dial digits will be prefixed assuming (in the default) they are dialed from Bell Laboratories at MH. These default attributes may be changed by the user at the node called the PREROOT (see Fig. 4), first encountered when the file is initially opened at the beginning of the access program. Functions at the PREROOT allow the user to change his default location or add another phone number temporarily to those listed in the header, or allow a new user to identify himself.

Each user is assigned a computer user-identification and a corresponding file system "home" directory. The user's private access tree is expected to be found in this directory along with other pertinent files like a personal directory, a mailbox, and a personal calendar. Besides each private tree-file, there is a collection of "public" tree files, any of which may be accessed by any user. The real users have access structures in which only real or imminent functions are accessible. The running example throughout this paper is such a tree. One pseudo-user, called "demo", has indicated access to named functions that are not implemented but from which provocative demonstrations have been given.

3.2.3 Program description

The structure (also in the sense of Michael Jackson) of the host-resident access program is illustrated in Fig. 5. The uppermost level shows the initial processing of the tree-file, the initial labeling of the buttons, and an iteration of processing "button pushes". Each button push is processed by reading the button identification message from the teleterminal, checking the validity of the message and its "type," executing the corresponding command with respect to current "modes," traversing the tree to the successor-node, and displaying the button labels pertaining to that node. The formats of various messages to and from the teleterminal are discussed in the next subsection. Command "modes" and "types" are briefly described.

The lock mode and the explain mode are program states that apply to the imminent button push but which are manipulated by a previous button push. The lock mode disables most commands and is a means for providing privacy protection: it is toggled by the CHANGELOCK command (see Fig. 4), which prompts for a password when "unlocking." The explain mode is enabled by the EXPLAIN command (see Fig. 4) and causes the program to substitute an explanation of the subsequent button push for the regular action of that button push.

A very brief description of each of the currently implemented commands is tabularized in Fig. 6. These commands are usually requested by the user through the operation of an appropriately labeled button. However, as another "type" of request, unprompted

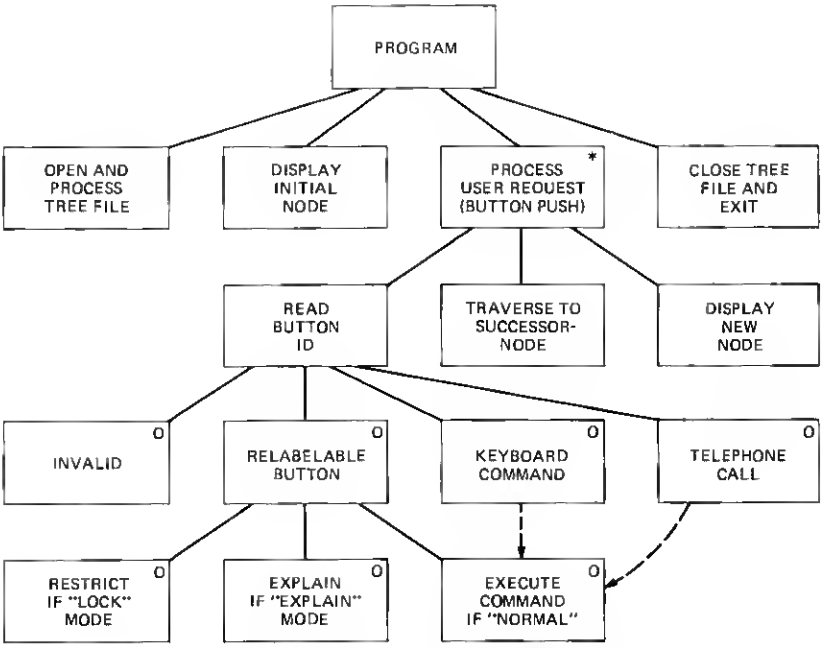

Fig. 5—Structure diagram of the access program.

keyboard entries are also interpreted as user-defined requests for specific commands. For example, unprompted numeric key entries are interpreted as telephone numbers.

Examination of Figs. 2, 4, and 6 will aid the reader in correlating the data file with the program. Several button pushes will be illustrated here, with more in the scenario presented in Section IV. In Fig. 4, the line beginning with "Personal Asst" (in the ROOT record-group) illustrates that **Personal Asst** is the label for button number eight on the root node (see Fig. 2). It is further seen from that record that when this button is selected by the user, the default TRAVERSE command is executed by the program and the tree traverses to the node whose record group is named PERSASST. Every command concludes with a traversal to the successor node; the TRAVERSE command does nothing else before that. The TRAVERSE command (see Fig. 6) requires no other data, and so none are present in the miscellaneous field of the record. The PERSASST record group is included in Fig. 4 and it corresponds to the button labels on the "Personal Assistant" node shown here in Fig. 7.

The line beginning with "System" on Fig. 4 illustrates that **System** is the label for button number ten on the root node (Fig. 2). When this button is selected by the user, the RUNSCROLL command is executed

COMMAND	DESCRIPTION
BACKRESTART	backup or restart in directory
CAL2MONTH	special interface for two-month calendar
CALENDAR	special interface for general calendar
CALL	call from button label
CALL2LINE	call from two-line button label
CALLLOCAL	call with local prefix
CALLPOTS	"POTS" call
CALLPREV	call party previously called
CALLPRFX	call with labeled tie-line prefix
CALLTRAV	"POTS" call or traverse in directory
CALLTREE	call from temporary tree
CATFILE	cat indicated file and exit or traverse
CHANGECALLTRAV	toggle "call/traverse" mode in directory
CHANGELOCK	toggle "lock" mode
DIRCATEGORY	select directory category
DIRINSTALL	proceed with personal directory installation
DIRTREE	make temporary tree from temporary directory
DUMMYCALL	dummy version of CALL
EXITENTRY	traverse to TREE with selected directory entry
EXITROOT	OTHERFILEEXIT or OTHERFILEROOT via mode
EXPLAIN	toggle "explain" mode
INSERTLABEL	insert button label into textual files
LEFTRIGHT	left/right traversal in directory
MAILRETURN	"detour" return to mail program
OTHERFILE	use other file instead of TREE
OTHERFILEEXIT	leave other file to exit point in TREE
OTHERFILEROOT	leave other file to TREE root
OTHERLOC	change default location
OTHERPHONE	enter "other number" for call-memos
READMAIL	special interface to read mail
REMINDER	special interface for reminder service
RUNBOTTOMLINE	use bottom line, create child, continue
RUNRETURN	clear & scroll, create child, label return button
RUNSCROLL	clear & scroll, create child, wait
SENDCALLMEMO	special interface to send a call-memo
SENDMAIL	send mail via *UNIX* software
STDCALLMEMO	special interface for standard call-memo
TEMPBACKUP	temporal backup
TEMPLATEBEGIN	initiate template creation
TEMPLATEFILL	fill in directory template
TEMPLATEINIT	initialize template from previous call
TEMPLATEMATCH	count or list template matches in directory
TRAVERSE	simple tree traversal to successor

Fig. 6—Current commands.

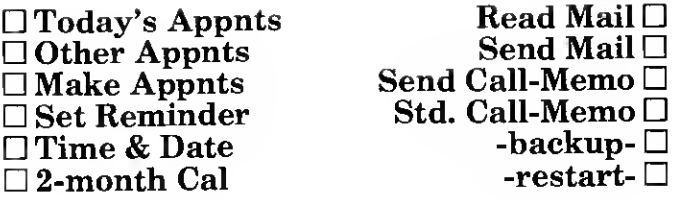

Fig. 7—The Personal Assistant node.

by the program and afterwards the tree traverses (by default) to the current node. In executing the RUNSCROLL command (see Fig. 6), the program clears the teleterminal of button labels and causes the indicated program (in this case, a *UNIX* program called the "shell"[14]) to execute from the teleterminal as a computer terminal. The screen now looks like that of a traditional terminal to the user, who is "talking" to the command interpreter of a supposedly familiar operating system. Upon termination of the shell, the access program gets control back and it executes the default traversal to the current, or root, node. In the "jargon" of the *UNIX* operating system, the activity is that the access program puts itself to sleep, and requests the spawning of a child process. The program expects to find the path to the particular child in the miscellaneous field of the record. When the user terminates the session with the child, it is killed and the parent (the access program) is awakened. The entire operation is perceived by the user as a leaf in that a function was requested, granted, and no traversal took place.

Changes to the data were discussed in the previous subsection. But not all changes to the access method can be implemented as simple "edits" of the access file. A new or changed command requires a program change and subsequent recompilation. However, the program has been structured specifically to simplify such changes. Command execution is implemented with a huge multiple branch. The C code in each branch is functionally decomposed. It should be relatively easy for a neophyte programmer (not our unsophisticated user, however) to modify this part of the program. A third kind of change is the unforeseen "nasty" kind of program change that requires a higher level of skill in the person making the change.

The user is expected to "log in" to his own account on the host from his teleterminal, and then to cause the access program to execute from his home directory. The program runs as a "shell" through which other programs, including the shell and other application programs, are invoked. It is planned for the invocation of the access tree program to be placed in the user's profile to facilitate "log in."

The internal firmware was designed so that communication between the user and the application software can proceed with a minimum of teleterminal-related details in the application programs. The areas of concern were system conventions imposed by the operating system, which impact the human interface to the application software and the potential for application software dependence on the detailed characteristics of the teleterminal. While the software was specifically implemented in the environment of the *UNIX* operating system, and the style of the paper assumes that, steps were taken in the design of the internal firmware to avoid assumptions about the specific operating system.

Operating systems establish conventions to control terminal I/O. Most systems usurp characters from the ASCII set for line deletion, character deletion, and interruption of program execution. In the *UNIX* operating system, the default symbols are: '@' to erase a line, '#' to erase a character, and DEL to interrupt program execution. Because our goal was to provide a consistent human interface independent of any operating system idiosyncrasies, the teleterminal allows the application to down-load the codes that should be used for these functions. Extra keys on the keyboard are provided for the erase-character, erase-line, and program-interrupt functions. Since conditions arise when a user may want to enter one of the special characters, the teleterminal will automatically prefix these characters with a host-specified escape-character (back-slash, by default, in the *UNIX* operating system).

Minimizing software dependence on the details of the teleterminal is achieved by defining a high-level interface to the functions:

(*i*) Labeling re-labelable buttons

(*ii*) Dialing

(*iii*) Forcing output to be placed on a single line of the display device.

By and large these functions are sufficient to allow all the tree-oriented software to execute on terminals that provide some form of relabelable buttons. Besides the use of a CRT to label buttons adjacent to the screen, alternatives are a button with embedded light-emitting diodes (LEDs) or a CRT with a touch-sensitive screen. Control of the screen is typically achieved by the host software through print statements. The teleterminal-specific details, such as the actual character strings defining control functions, can be isolated in one file. Using the macro facility of the C language, these control messages can be made to look like function calls. For example, a macro for labeling buttons may be defined such that the statement:

```
button_label(btn_num,"Prefix Call");
```

would expand to the C statement:

```
printf("%c%c%s\n",ESC,'a'-1+btn_num,"Prefix Call");
```

When executed (assuming btn_num is equal to 2) this statement results in the string

ESC b Prefix Call CR NL

being transmitted to the teleterminal, which causes the second button on the left-hand side of the screen to be cleared, then labeled with "Prefix Call". Similar macros may be defined for other primitives.

IV. A SCENARIO

The functionality of the tree-like access method is demonstrated by a scenario wherein **A** calls **B**, but **B** is either busy or not home: **A** leaves **B** a message to return the call; then **B** becomes available, reads the message, and returns the call to **A**. We (a user named "us") are **A**, Susan is **B**, and the scenario begins at the root node (Fig. 2). A monologue for introducing a new user to the teleterminal, its services, and tree-like access is appropriate for the unsophisticated user before presenting this scenario. Such a monologue is illustrated in Ref. 5.

We select the **Susan** button and place a call to Susan, the department secretary. Customization of the tree-file is demonstrated by the presence of the **Susan** button in the root, indicating that this call is frequently made; and also by the button text itself: **Susan** is a "friendlier" label than initials and last name (like a typical directory entry) or **secretary**. As seen in Fig. 4, the miscellaneous field of the "Susan" record in the tree-file contains her telephone number and her computer user identification. The CALL command (see Fig. 6) causes the telephone number to be dialed and the id to be stored for possible subsequent use. The tree is traversed to the **Personal Assistant** (Fig. 7).

As an aside, this last traversal exemplifies two points already made. The user, in customizing his own tree-file, determines the successor node of *branches* like **Susan**. There are three appropriate choices:

(*i*) Traversal to the current node emphasizes the leaf nature of such a botton and, often, if such a call fails to complete, a "related" call (that is, to someone on the same node) may be the logical subsequent event.

(*ii*) Traversal to the root gives the perception of "resetting" the teleterminal for subsequent use.

(*iii*) Traversal to the **Personal Assistant** is appropriate if calendar and message functions are likely to be subsequently used.

With the use of **restart** and **backup** buttons, the decision is not all that critical. The second point that is exemplified is the fact that the access tree is not, strictly speaking, a tree: two branches to the **Personal Assistant** have already been demonstrated.

Continuing with the scenario, let **Susan** be busy and let us send her a "Call-Memo" requesting that she call us back. There are two "Call-Memo" buttons on the **Personal Asst** node (Fig. 7): **Send Call-Memo** prompts the user through the creation of a general call-memo and **Std Call-Memo** immediately sends a standard "I called you—Please call me back" call-memo. This standard message is created once by, or for, the user and is available in the user's directory along with a host of other handy files, like a personal directory and a personal calendar. We select **Std Call-Memo** and we are prompted on the bottom line of our teleterminal as shown here in Fig. 8.

The program can't be sure that the call-memo is intended for the person last called, but that is the default operation. If the user wishes to change the default, he identifies the desired user (typing is echoed after the colon on the bottom line). We simply type the "CR" key to continue the default operation and the scenario proceeds. The program responds on the bottom line as shown here in Fig. 9.

Proceeding to the second half of the scenario, **Susan** is tracked as she returns the call. Her first action is to read her mail. From the root (assuming that her tree-file resembles ours), she selects **Personal Asst** and then **Read Mail**. A special mail program is invoked that interfaces the mail program provided in the *UNIX* operating system

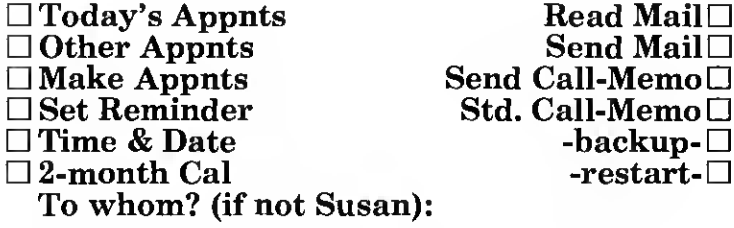

Fig. 8—The **Personal Assistant** node with **Std Call-Memo** prompt.

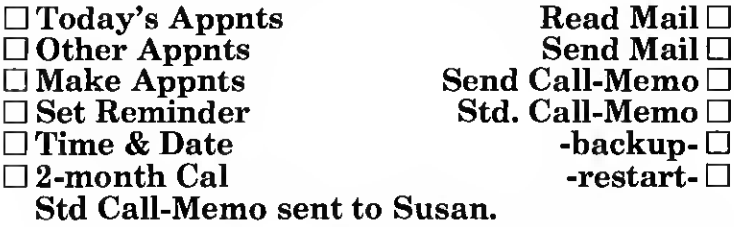

Fig. 9—The **Personal Assistant** node with **Std Call-Memo** response.

to the teleterminal. If our call-memo is her only new message, the screen appears as shown here in Fig. 10.
After selecting our message, the call-memo is scrolled onto the screen as shown here in Fig. 11.

There are a number of different responses to such a message: the choice of action is made after pressing the **process msg** button, as shown in Fig. 12. In our scenario, Susan elects to return the call and selects **Return Call**. Our telephone number is extracted from the call-memo by the program, the mnemonic prefix **"MH"** is appropriately translated into the correct dial prefix, depending on the location of Susan's teleterminal, and the call is made. The screen returns to the call-memo (Fig. 11) from which point the message would probably be **Thrown Out**, ending the scenario.

User perception and the importance of cosmetics cannot be overemphasized. The reader **must** be aware that the demonstration of this very scenario on an actual teleterminal is many times more appealing than a description of the demonstration in such a paper as this. Many

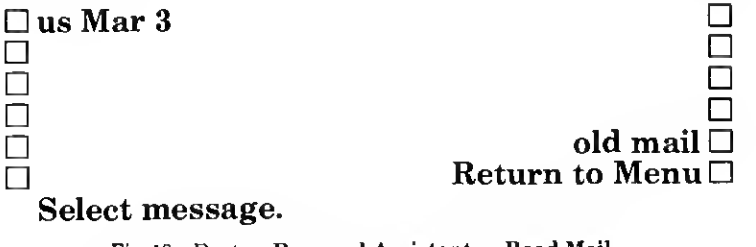

Fig. 10—Root → **Personal Assistant** → **Read Mail.**

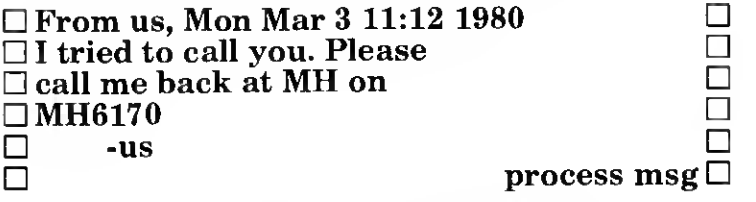

Fig. 11—The Standard Call-Memo.

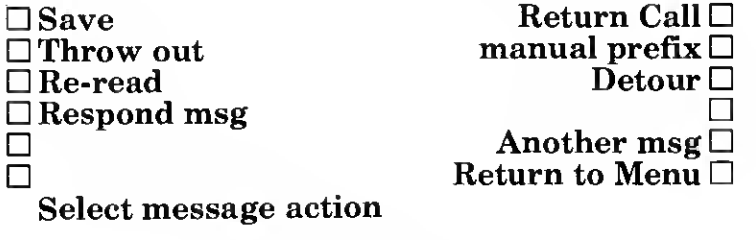

Fig. 12—The Message Action menu.

other functions have been implemented but the interest of brevity prevents any description. These include a personal calendar (see the first three buttons on Fig. 7), organizational and alphabetic directories, special-purpose directories (such as emergency numbers, dial-a-"x", and assistance), a personal telephone directory, manual selection of dial prefixes, and a food applications demonstration. In addition, the tree provides access to standard *UNIX* programs supplying facilities like a monthly calendar, time and date, a desk calculator simulation, and a large selection of computer games. Furthermore, accesses to a voice storage system and a dial-up dictation system have been implemented that present a highly satisfactory human interface.

V. FUTURE DIRECTIONS

The functional split between the teleterminal and the host has provided a flexible and efficient interface for experimenting with new applications. We have "ported" the application software from a Digital Equipment PDP 11/45 system using the MERT[15] operating system to a Tandem computer using the Guardian[16] operating system. The human interface remained the same even though the I/O characteristics of the two systems are quite different.

Our experience exposed several areas where the current system is deficient. In the human interface, the teleterminal provides special keys for delete line, delete character, and program interrupt. Two additional special keys labeled "end of file" and "program abort" are needed to provide a completely operating-system-independent interface. Keys with the permanent labels "explain," "backup," and "restart," would improve the human interface to the access tree by freeing three of the relabelable buttons for other functions. An extension of the screen management allowing an arbitrary screen split between button labels and scrolled text would improve the human interface by allowing communications in excess of 32 characters.

The interface between the teleterminal and application programs could be improved by having the teleterminal emit a special button status message when the telephone goes off-hook. This would allow applications to respond directly to dial requests, eliminating a button push under some circumstances. A button label frame buffer capable of retaining about 30 button labels coupled with a special "more" key would remove the teleterminal-specific restriction that no more than twelve buttons can be labeled at one time.

The current teleterminal interface to the host supports a single character-oriented data stream. Application software must take special care to ensure that a message signaling the occurrence of an asynchronous event does not get interspersed with other messages. For example, if an electronic mail application wanted to inform the user of the

arrival of a message, the application software would have to coordinate an attempt to write the message "you have mail" with all other programs that might have reason to write to the instrument. This responsibility belongs in the network and in the teleterminal.

The solution requires the network to support some form of data multiplexing. For teleterminal applications, the network must support a control channel and several data cbannels. The control channel is a message-oriented channel used to establish the switch between data channels. The network must ensure that control messages are atomic, delivered error free, and are flow controlled. In order to ensure that control messages remain synchronized with the data channels, it is advisable that the control channel be implemented as a subchannel on a multiplexed data stream. The teleterminal would require enhancements to conform to the message protocol of the control channel and to route data over the appropriate data channel. It is sufficient that the teleterminal be able to maintain several data connections but only be able to receive (and send) data over one channel at a time.

For a network with this form of data multiplexing, asynchronous events, such as the arrival of electronic mail, could be implemented by sending an "alert" message to the teleterminal over the control channel. The teleterminal would respond by flashing a button with the **alert** label. The user would eventually respond to the alert by pressing the **alert** button, causing the teleterminal to send a "hold" status message over the current active data channel, switch to the mail data channel, and send a "proceed" status message over the mail channel. The mail program could then take control of the screen. To return to the program interrupted by the mail function, either the user would press a permanently-labeled **resume** buttom or the mail program would automatically return by sending an appropriate control message to the teleterminal. In either case, the teleterminal would send a status message over the previous data channel. The application program that had been put on hold would take over the screen by rewriting the last frame before the interruption.

A new design of the teleterminal has been recently completed and it incorporates many of the features described. Cosmetically, the screen is logically the same size as that of a conventional computer terminal and the keyboard is large enough for touch-typing (see Fig. 13). Thus, the set of services appropriately addressed is expanded from "enhanced telephony" to include information management and office automation.[17] Functionally, the new teleterminal provides a telephone interface like that of a traditional 6-button key telephone, thus allowing easy interface to call directors and speakerphones. The internal processor is a "microcomputer system" running under a *UNIX*-like operating system.[18] At this time it simply emulates the functions of the old

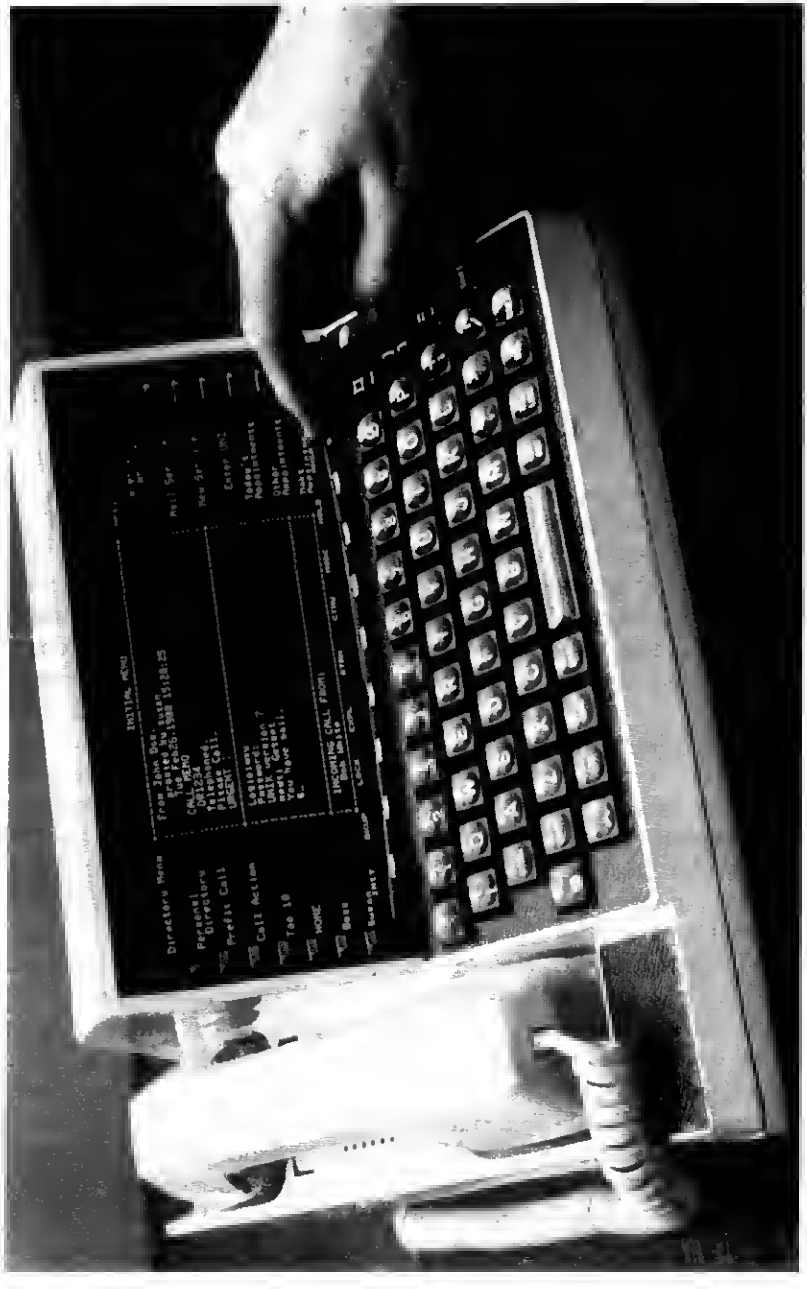

Fig. 13—Redesigned teleterminal.

internal software; however, many more elaborate functions are planned.

One of these is the capacity for a multiplicity of message "windows," each supporting one of the independent data connections mentioned above. Investigations into combining voice and data onto one digital channel are proceeding.[19] New applications become feasible once the teleterminal is connected to an intelligent switching office: for example, displaying the number of a calling party and a variety of transfer and screening functions. Many new capabilities, and improvements to existing capabilities, have been suggested by the teleterminal users and these are gradually being added.

VI. CONCLUSION

This paper has described the software strategy behind the construction of a research tool called a teleterminal. The realization of this strategy lies in a software architecture whereby intelligence is distributed between the teleterminal's internal processor and a host computer. The programs resident in each of these locations, and their intercommunication, have been discussed.

More in the abstract, another part of this strategy is an access method whereby the user of a future teleterminal can conveniently interface to the abundance of applications that could be available. General concepts have been presented and a first implementation has been described from the user's viewpoint. The proposed access method has been found highly acceptable by colleagues and acquaintances but formal testing[20,21] on the general public has only just begun. This is ongoing work and this paper represents a "snapshot in time" that is already slightly out of date. The first working teleterminal was demonstrated internally in September of 1978. Since that time three dozen teleterminals have been constructed and a user group assembled. The software has undergone innumerable changes, and the newer "model" was designed and constructed.

Acknowledgments are owed to the following colleagues for their contributions to the software effort of the project: Bob Anderson for assistance with the internal software, Martin Sturzenbecker for the special mail program and a filter that enables the access program to work at a regular terminal, Ron Gordon for the tree "compiler" and the definition of the "high-level" data language illustrated in Fig. 4, Ron and Martin for a "Food Applications" program that clearly illustrates the value of organizing "atomic functions" into "generic capabilities," Misha Buric for the personal calendar program, Al Usas for support with the "*UNIX*-Tandem" environment, and Bob Allen and Donna Zanolla for their human factors testing.

REFERENCES

1. D. W. Hagelbarger, R. V. Anderson, and P. S. Kubik, "Experiments with Teleterminals," *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, 1981, pp. F2.1.1-5.
2. D. W. Hagelbarger, "Experiments with Teleterminals," *B.S.T.J.*, this issue.
3. G. D. Bergland, "An Experimental Telecommunications Test Bed," *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, 1981, pp. F2.3.1-5.
4. R. W. Lucky, "A Flexible Experimental Digital Switching Office," *Proc. 1978 Int. Zurich Seminar on Digital Commun.*, Zurich, Switzerland, 1978.
5. R. A. Thompson, "Accessing Experimental Telecommunications Services," *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, 1981, pp. F2.2.1-5.
6. G. Cohen, *The Psychology of Cognition*, New York: Academic Press, 1977.
7. G. T. Uber, et al., "The Organization and Formating of Hierarchical Displays for the Online Input of Data," *Proc. Fall Joint Computer Conf.*, 1968.
8. D. L. McCracken and G. G. Robertson, "Editing Tools for ZOG, a Highly Interactive Man-Machine Interface," *Proc. Int. Conf. Commun.*, Boston, MA, 1979.
9. M. Behzad and G. Chartrand, *Introduction to the Theory of Graphs*, Boston, MA: Allyn and Bacon, 1971.
10. "The UNIX Time-Sharing System," *57*, Part 2, a dedicated special issue of the *B.S.T.J.* (July-August 1978).
11. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Englewood Cliffs, NJ: Prentice-Hall, 1978.
12. R. D. Gordon and D. L. Smith, "An Access Tree Editor," *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, 1981, pp. F2.7.1-5.
13. M. A. Jackson, *Principles of Program Design*, New York: Academic Press, 1975.
14. S. R. Bourne, "The UNIX Shell," *B.S.T.J., 57* (July-August 1978), pp. 1971–90.
15. H. Lycklama and D. L. Bayer, "The MERT Operating System," *B.S.T.J., 57* (July–August 1978), pp. 2049–86.
16. *Tandem 16, Guardian™ Operating Manual.* Tandem Computers Inc., 1980.
17. R. N. Klapman, "Enhanced Communications in an Executive Office," *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, 1981, pp. F2.4.1-5.
18. W. M. Schell, "Control Software for an Experimental Teleterminal," *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, 1981, pp. F2.8.1-5.
19. R. A. Thompson, "An Experimental User-Resident Communications Controller Supporting Sub-Rate Circuit-Switched Service," *Proc. Int. Symp. on Subscriber Loops and Services*, Munich, Germany, 1980, pp. 68–71, and *the IEEE Trans. Commun., COM-30*, Number 6 (June 1982), pp. 1399–408.
20. R. B. Allen, "Cognitive Factors in the Use of Menus and Trees: An Experiment," *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, 1981, pp. F2.5.1-5.
21. R. A. Thompson, "User's Perceptions with Experimental Services and Terminals," *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, 1981, pp. F2.6.1-5.